Linear Reversible Circuit Synthesis and Optimization

2021-01-20

Presenter:黃彥翔

National Taiwan University

Taipei 106, Taiwan

Outline

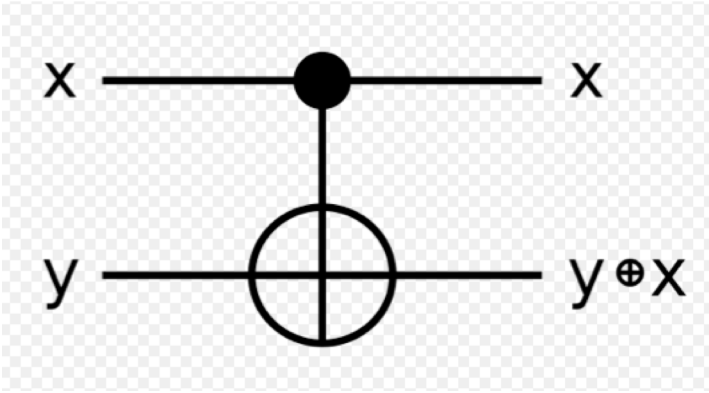
- Introduction and previous work
- Implementation and Contribution
- Experimental Results
- Observation and Discussion

Outline

- Introduction and previous work
- Implementation and Contribution
- Experimental Results
- Observation and Discussion

C-NOT gates and linear reversible circuit

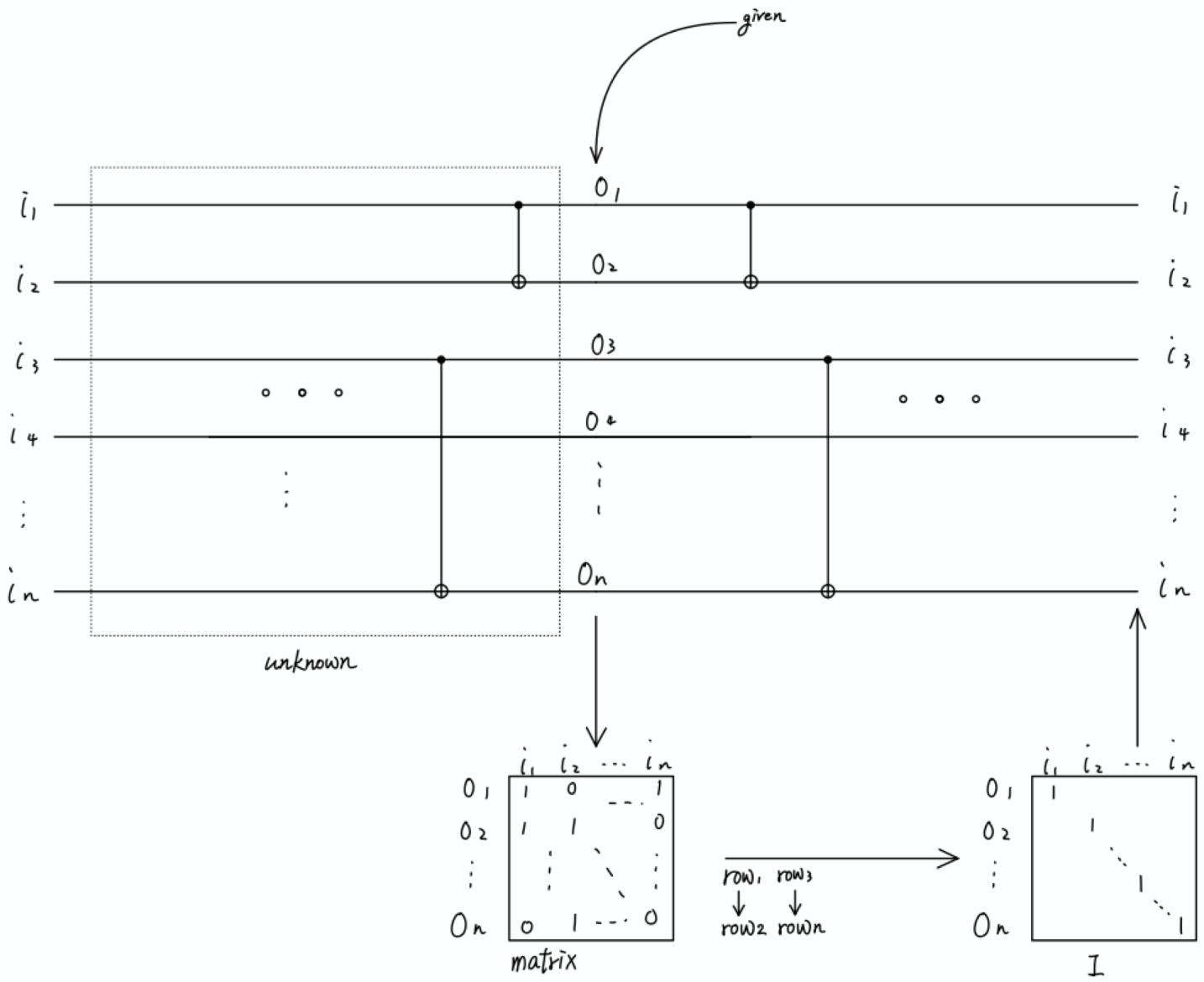
- A circuit consists of only C-NOT gates is linear and reversible.
  - It is linear because of the output functions are linear functions
  - It is reversible because there is a simple way of construct a reverse function: reverse the sequence of the C-NOT gates



Gaussian elimination and synthesis

- Given the output functions, we can use Gaussian elimination to synthesis a circuit
  - A row operation is the addition of one row to another. In the view of a circuit, it is equivalent to the addition of one output to another.
  - Thus, if we can use Gaussian elimination to make the output matrix become a identity matrix, we can obtain a circuit with inverse function.
  - With the reverse circuit, we can obtain the circuit with the original function.

Gaussian elimination and synthesis



Gaussian elimination and synthesis

- Defect:
  - Complexity is *O(n^2)*, however the complexity of reducing a matrix to identity using row operations is proven as *O(n^2/log(n))*

Previous Work

- Ketan N. Patel, Igor L. Markov and John P. Hayes has proposed a algorithm in 2003, with complexity of *O(n^2/log(n))* to reduce a matrix to a identity matrix using only row operation if possible

Previous Work

- Proposed algorithm:
  - Lwr_CNOT_Synth:

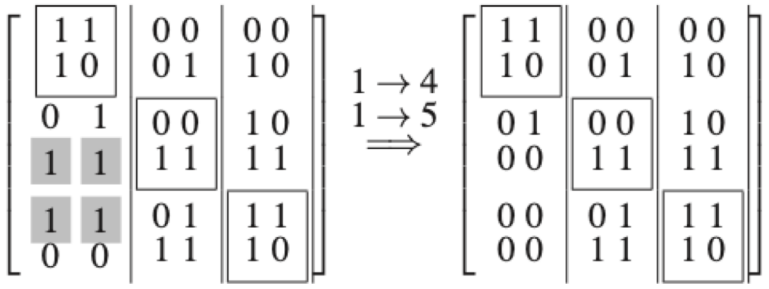
  Partition matrix into ”sections”, which consists *m* columns.

  For each sections, do these three steps:

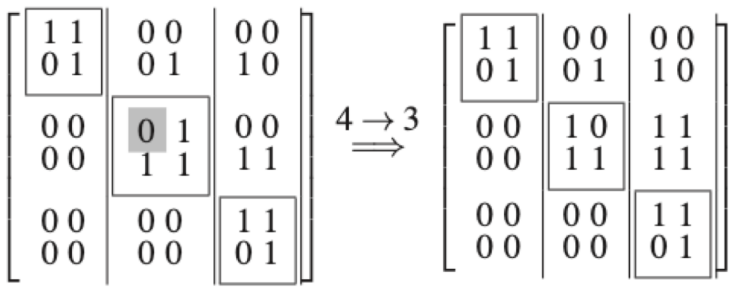
  step A: eliminate duplicate sub-rows

  step B: make the section diagonal

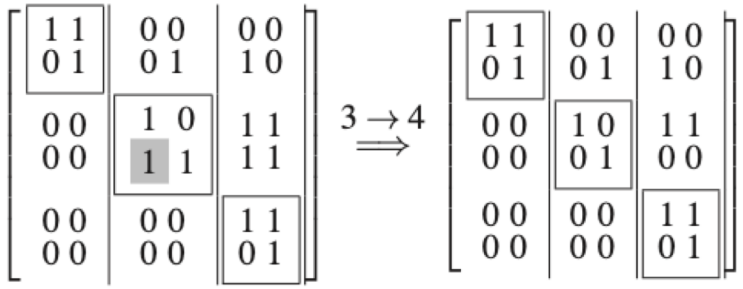
  step C: remove remaining ones(same approach as Gaussian elimination)



A



B



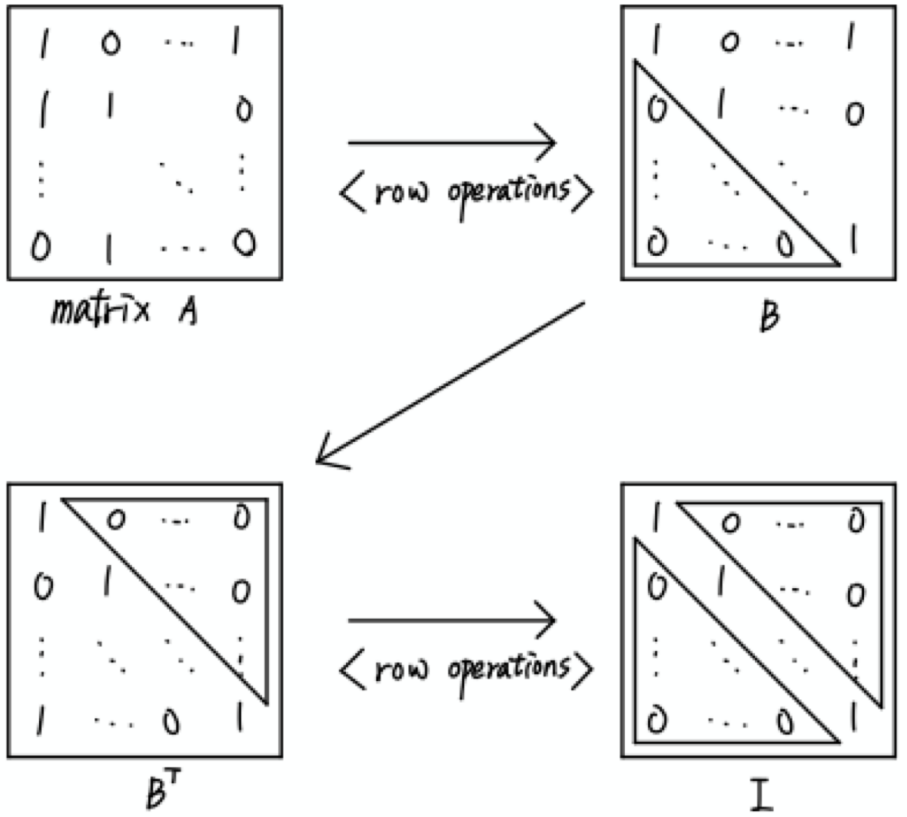
C

Previous Work

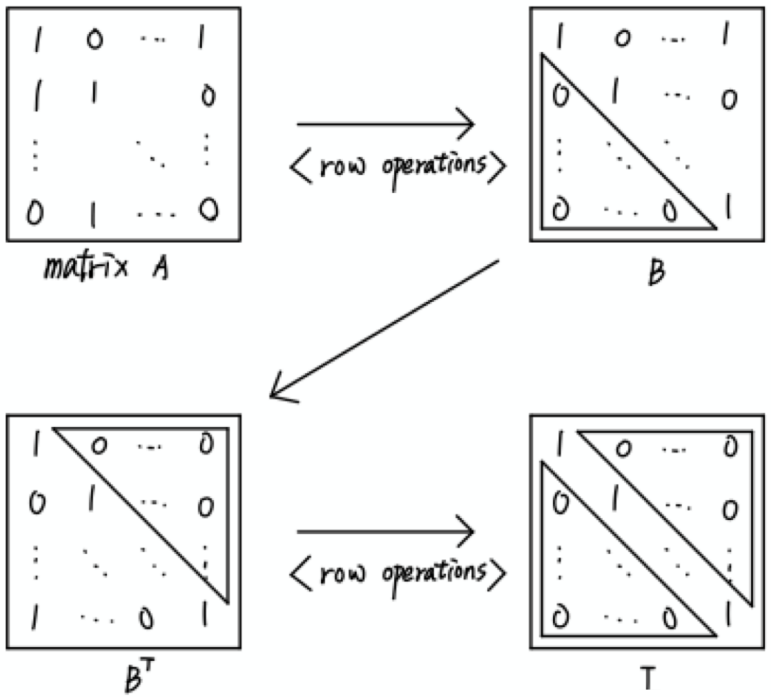
- Proposed algorithm:
  - CNOT_Synth:

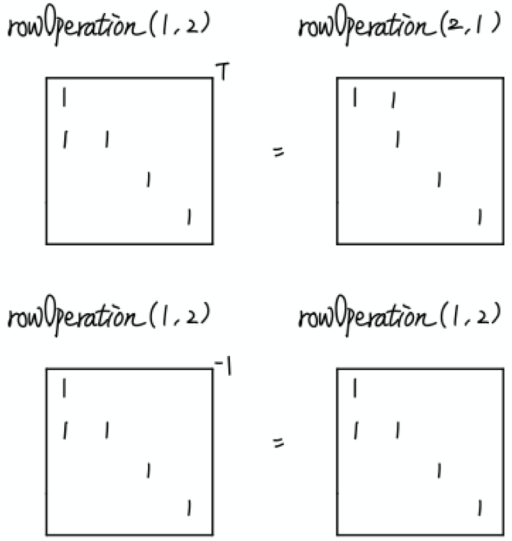
```
[circuit] = CNOT_Synth(A, n, m)
{
   // synthesize lower/upper triangular part
   [A,circuit_l] = Lwr_CNOT_Synth(A, n, m)
   A = transpose(A);
   [A,circuit_u] = Lwr_CNOT_Synth(A, n, m)

   // combine lower/upper triangular synthesis
   switch control/target of C-NOT gates in circuit_u;
   circuit = [reverse(circuit_u) | circuit_l];
}
```



Previous Work





$$R_{l_i} R_{l_{i-1}} \cdots R_{l_2} R_{l_1} A = B$$

Circuit_l

$$R_{U_j} R_{U_{j-1}} \cdots R_{U_2} R_{U_1} B^T = I$$

Circuit_l

$$\Rightarrow B R_{U_1}^T R_{U_2}^T \cdots R_{U_{j-1}}^T R_{U_j}^T = I$$

$$\Rightarrow B = R_{U_j}^T R_{U_{j-1}}^T \cdots R_{U_2}^T R_{U_1}^T I$$

$$\Rightarrow R_{l_i} R_{l_{i-1}} \cdots R_{l_2} R_{l_1} A = R_{U_j}^T R_{U_{j-1}}^T \cdots R_{U_2}^T R_{U_1}^T I$$

$$\Rightarrow A = R_{l_1} R_{l_2} \cdots R_{l_{i-1}} R_{l_i} R_{U_j}^T R_{U_{j-1}}^T \cdots R_{U_2}^T R_{U_1}^T I$$

```
switch control/target of C-NOT gates in circuit_u;
circuit = [reverse(circuit_u) | circuit_l];
```

Previous Work

- Complexity:

$$\text{total row ops} \le (n+m)\cdot\left\lceil\frac{n}{m}\right\rceil + n + 2\left\lceil\frac{n}{m}\right\rceil m\cdot(2^m+m)$$

$$\le \frac{n^2}{m} + n + n + m + n + 2n2^m + 2nm + 2m2^m + 2m^2$$

$$m = \alpha\log_2 n,$$

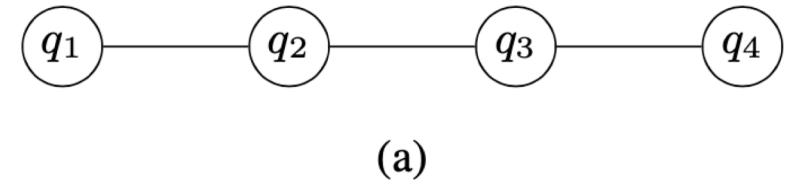
$$\text{total row ops} \le \frac{n^2}{\alpha\log_2 n} + 3n + \alpha\log_2 n + 2n^{1+\alpha} + 2n\alpha\log_2 n + 2\alpha\log_2 n\cdot n^{\alpha} + 2(\alpha\log_2 n)^2.$$

thus, in my implementation, I choose $\alpha = \frac{1}{2}$

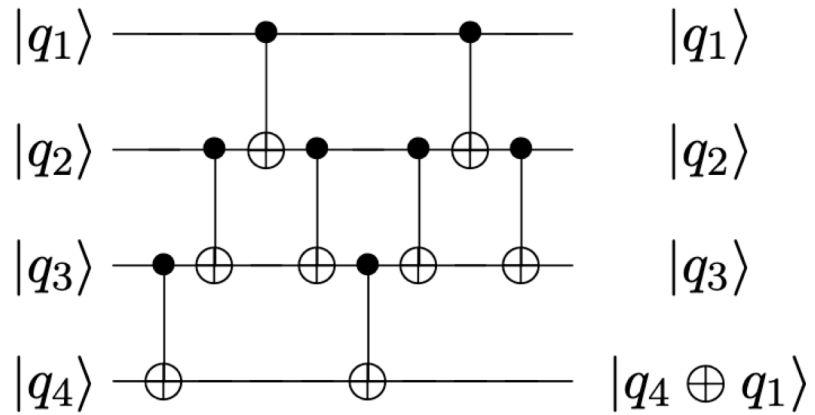
- Time complexity is *O(n^3/log(n))*, compare to standard Gaussian elimination's *O(n^3)*

Matrix Reducing with Connectivity

- Connectivity limits CNOT gate to be only apply on neighbor physical qubits
- Any row operation (or qubit addition) can be performed using the template, however it is very gate consuming.
- To use the method just mentioned, it increase the complexity from *O(n^2/log(n))* to *O(n^3/log(n))*
- A algorithm of complexity of *O(n^2)* is proposed.



(a)



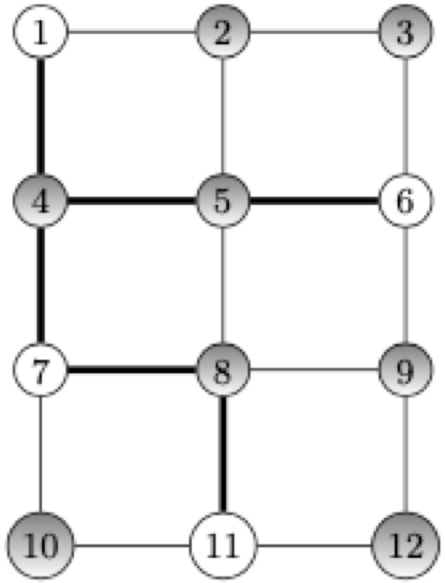
BeatriceNash,VladGheorghiu,MicheleMosca,
Quantum circuit optimizations for NISQ architectures,2020

Matrix Reducing with Connectivity

- Problem: Given a set of rows to be reduced, how to determine the row operation sequence with the least operations.

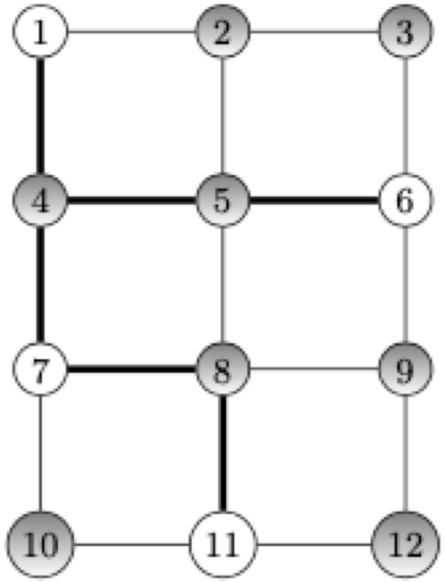
- The problem can be reduces to the Steiner tree problem on the connectivity graph.

- Some nodes need to be reduce(terminals),
  but the others didn't(Steiner nodes).
  A algorithm is proposed to handle this situation.



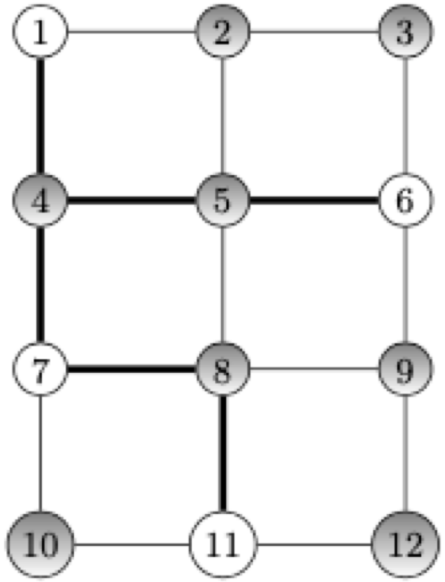
Proposed algorithm

- Construct a tree:
  - Perform BFS search from the terminals, when two terminals collide, combine them and treat them as a new terminal.(somewhat like a Multi-source Maze router)
  - Don’t use partitioning



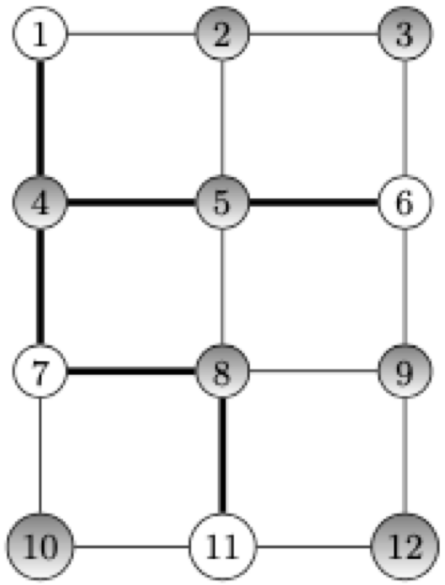
Proposed algorithm

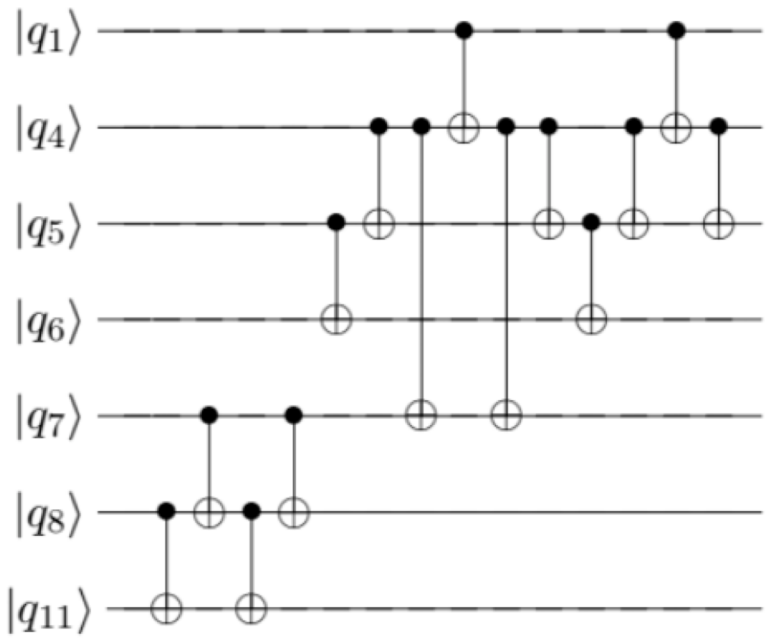
- Construct subtrees:
  - Perform BFS search from the root, when arriving a terminal, stop and build a new sub-tree for the terminal.
  - Thus, every tree has terminals at root and leaves.

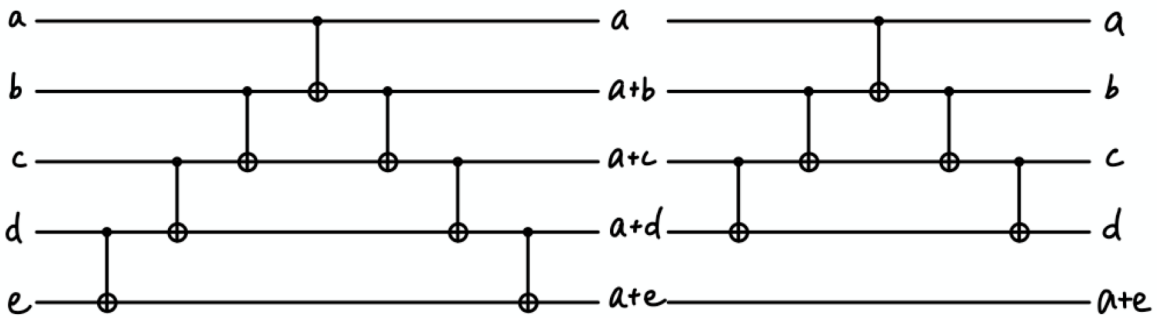


Proposed algorithm

- Construct circuit(before transpose):
  - Start from the last sub-tree constructed.
  - Perform DFS for each sub tree from the root. When visit a edge (u, v), append (u, v) into the CNOT-gate sequence *R*
  - *R' = R-R[j], R* = (R+R')/(gate with terminal as target)*
  - *Circuit = R+R'+R**

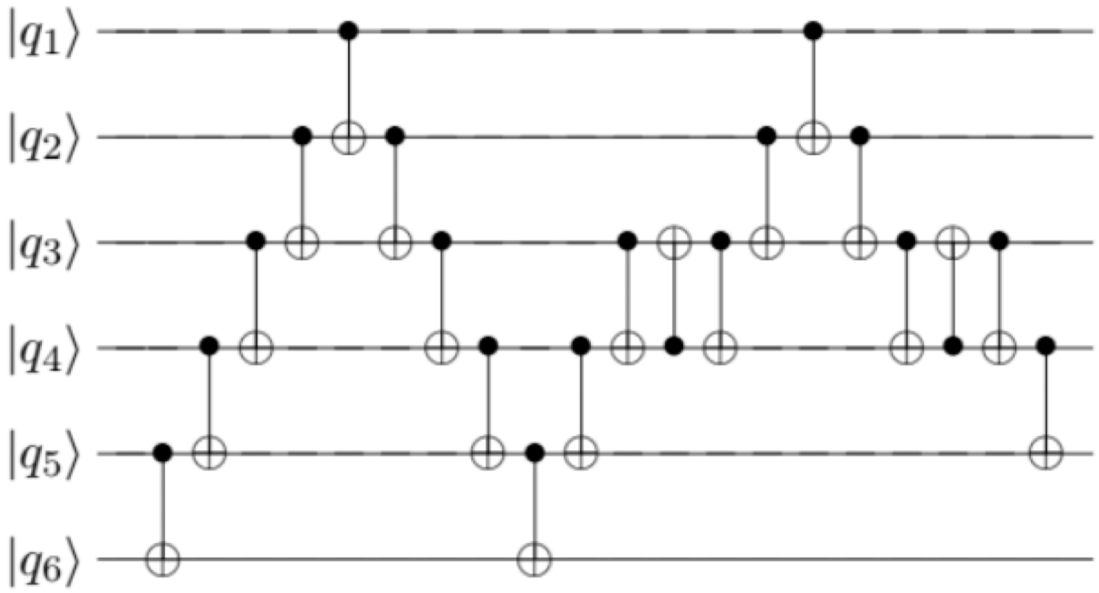


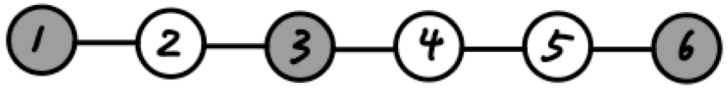




Proposed algorithm

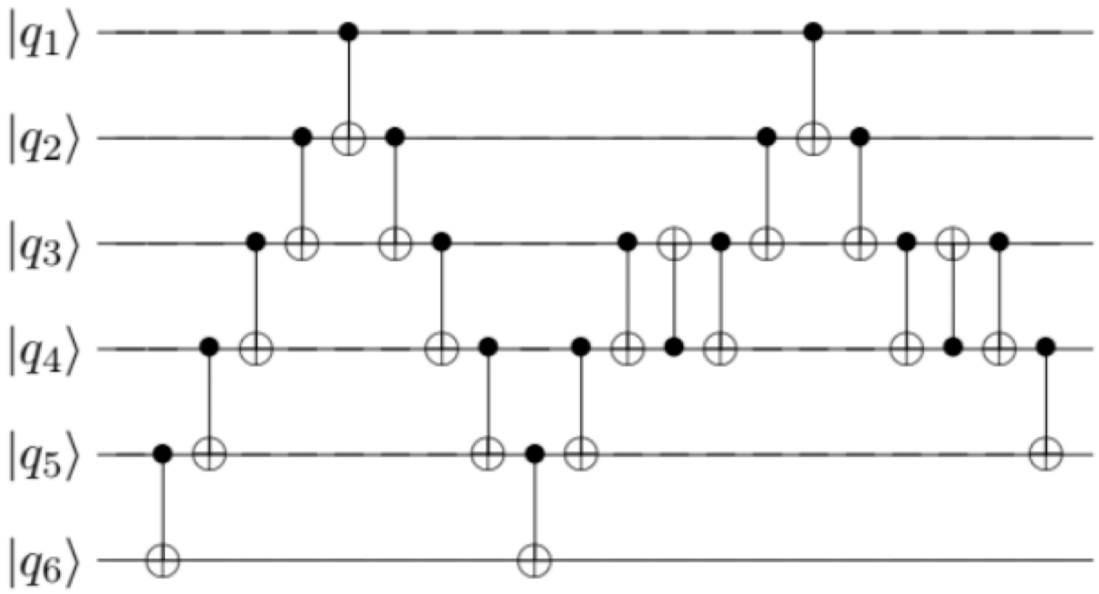
- Construct circuit(after transpose):
  - Don’t use the sub-trees. Since it may ruin the lower-triangular form.
  - Traverse the whole Steiner tree and build *R* and *R’* as usual.
  - Build R* the same as the step before transpose, however, perform a “swap gate” when arriving a terminal that is not a leaf.

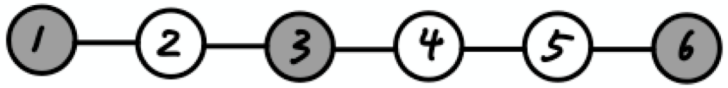




Complexity

- Each tree: #gate = *O(n)* time = *O(d*(|E|+|V|))*, where d is #terminal
- Overall: #gate = *O(n^2)* time = *O(n^2*(|E|+|V|))*





Outline

- Introduction and previous work
- Implementation and Contribution
- Experimental Results
- Observation and Discussion

Implementation and Contribution

- Language: c++17
- Os: ubuntu20.04

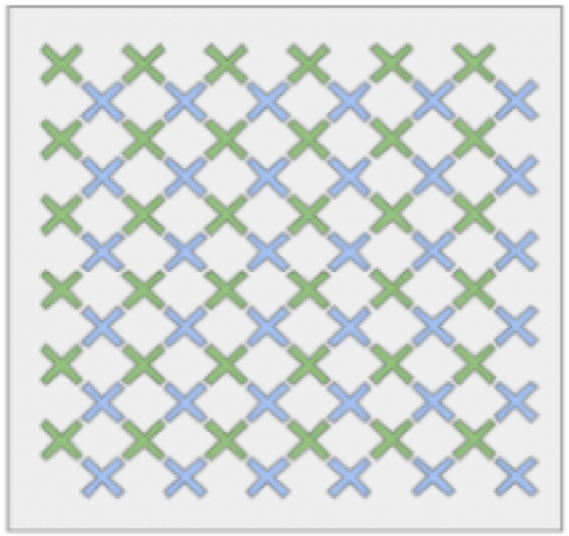
Implementation and Contribution

- For the circuit synthesis without connectivity, I had implement it and verify the gate complexity using some experiment, the result will be shown later.

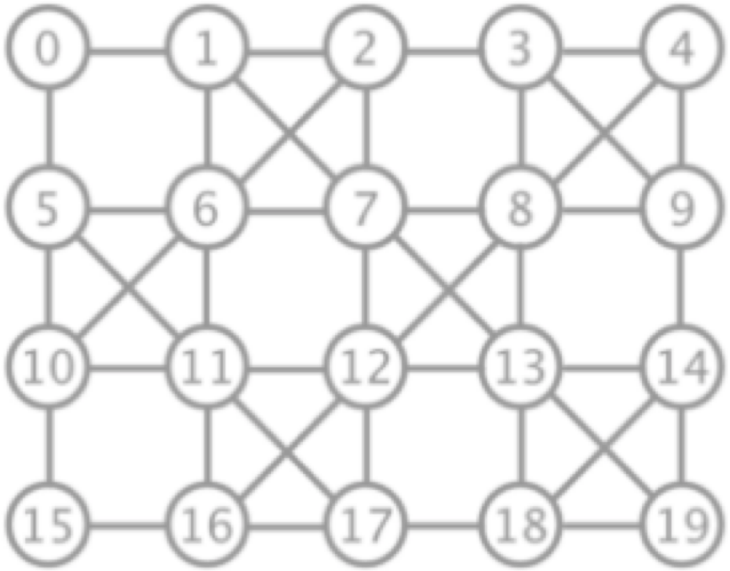
- For the circuit synthesis with connectivity,

Implementation and Contribution

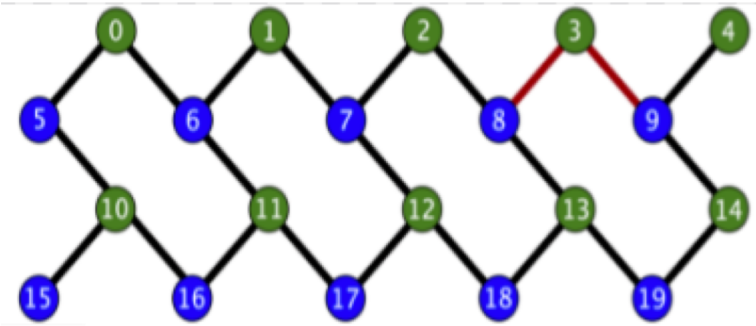
- For the circuit synthesis with connectivity, I found that the connectivity graph they used, such as Google's Bristlecone72, IBM's Tokyo20, and Rigetti's Acorn19, are not that complicated.
- There may be another good mays for build a Steiner tree, instead of applying a general but time consuming and inaccurate method.



Bristlecone72



IBM's Tokyo20



Rigetti's Acorn19

Implementation and Contribution

- If the connectivity graph rectangular, consists of grids, some algorithm for fast approximation of Steiner tree can be applied, such as FLUTE.

- With FLUTE, time complexity can be reduced to *O((d)*log(d)),* where d is #terminal

- And FLUTE has a really low wirelength($\sim 1\%$ with degree up to 500)

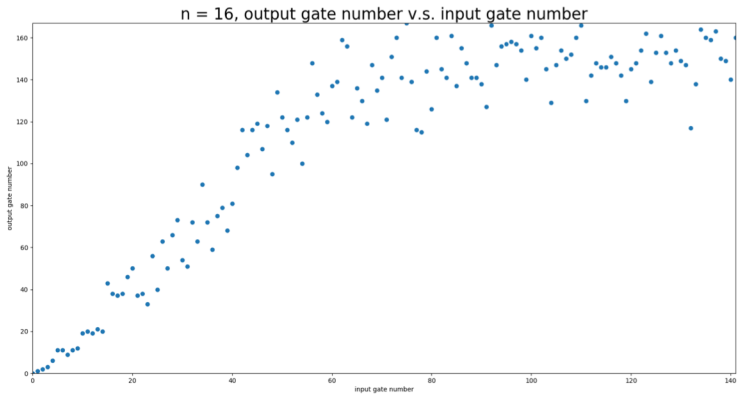
Implementation and Contribution

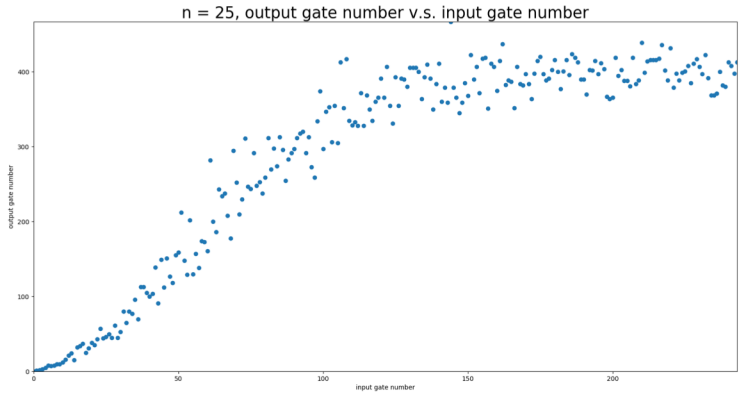
- Some visualization tools to draw the circuit and connectivity graph.

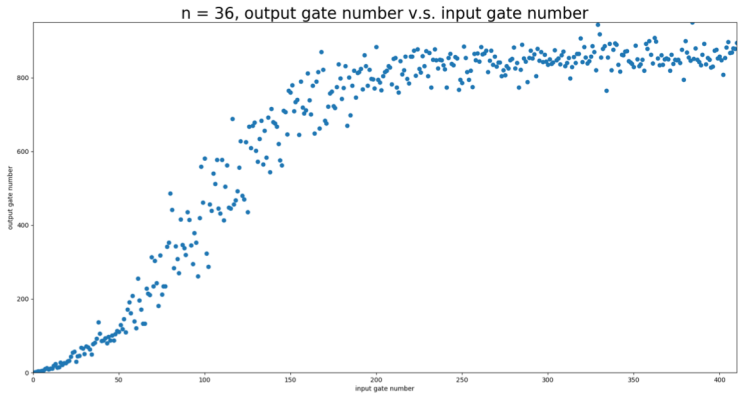
Outline

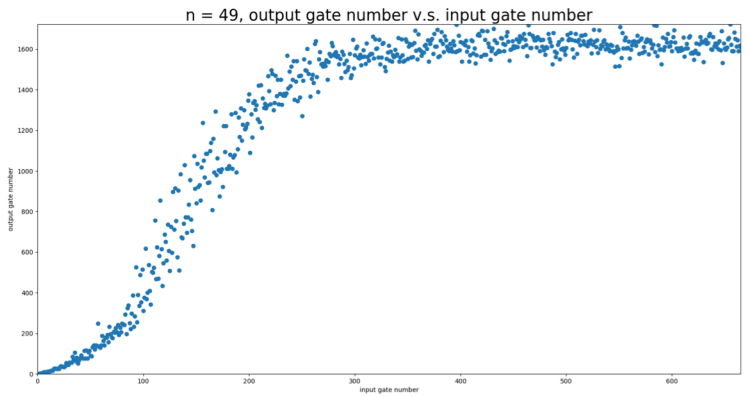
- Introduction and previous work
- Implementation and Contribution
- Experimental Results
- Observation and Discussion

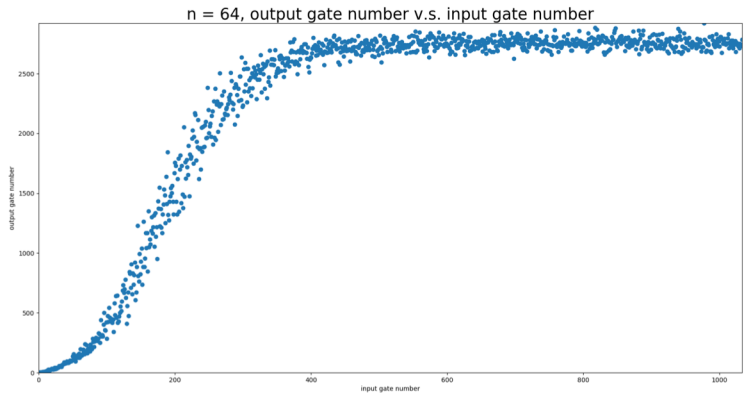
Experimental Results without connectivity

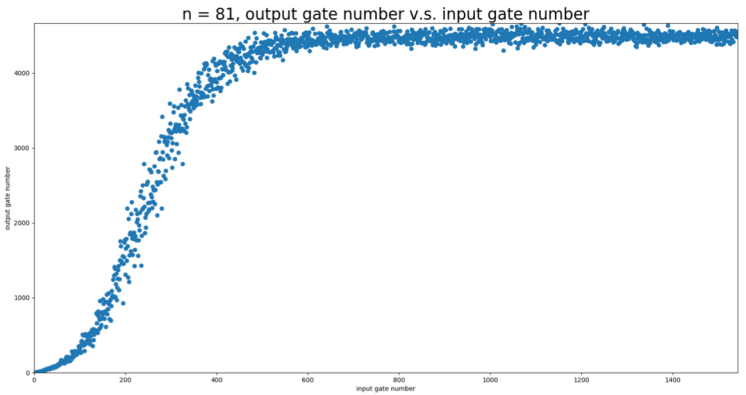






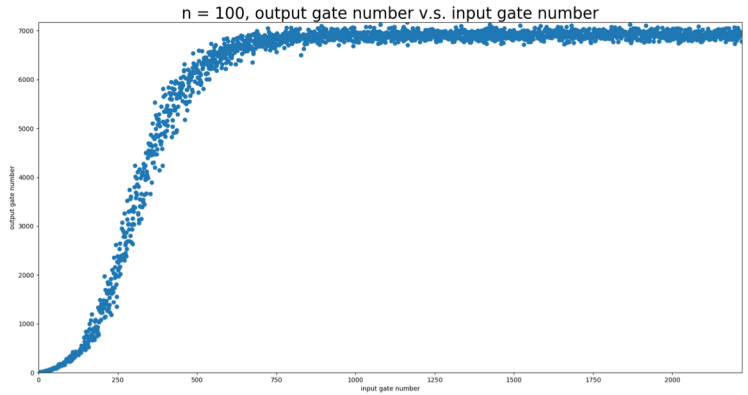


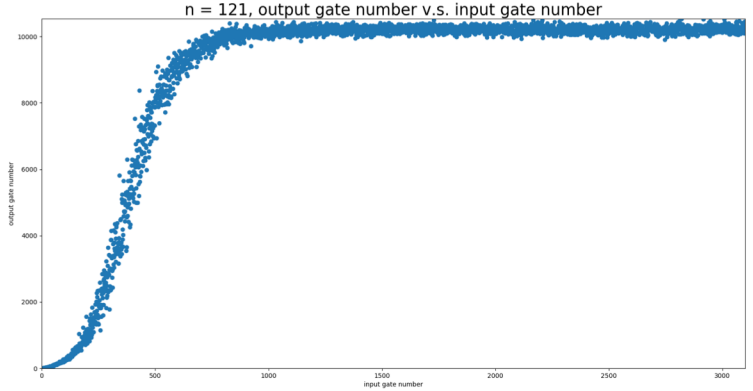


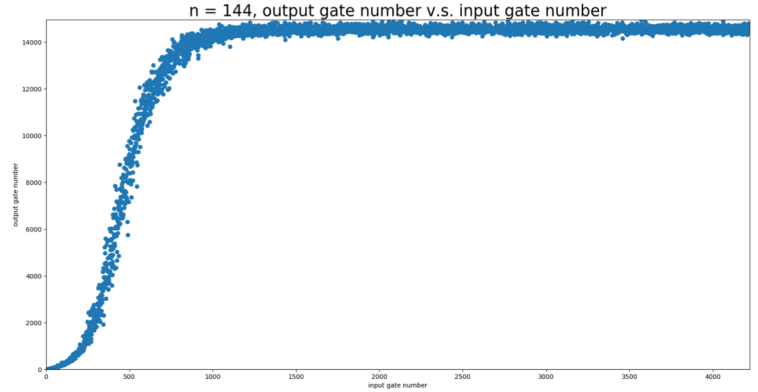


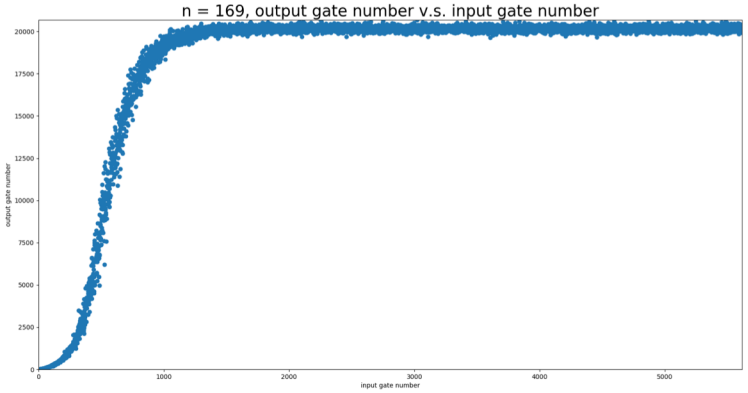
- Test circuit is generated by randomly generate C-NOT.

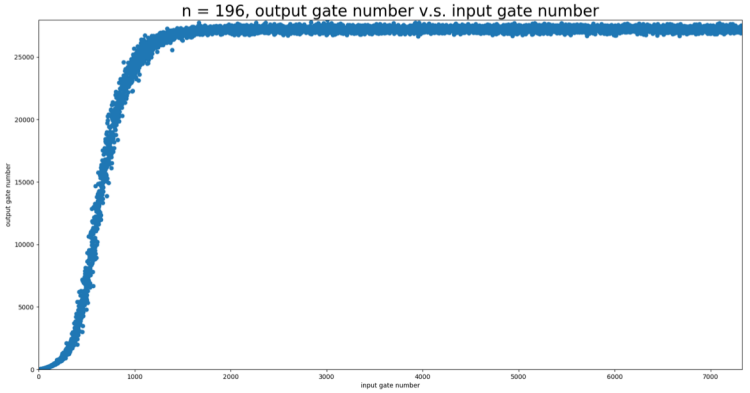
Experimental Results without connectivity

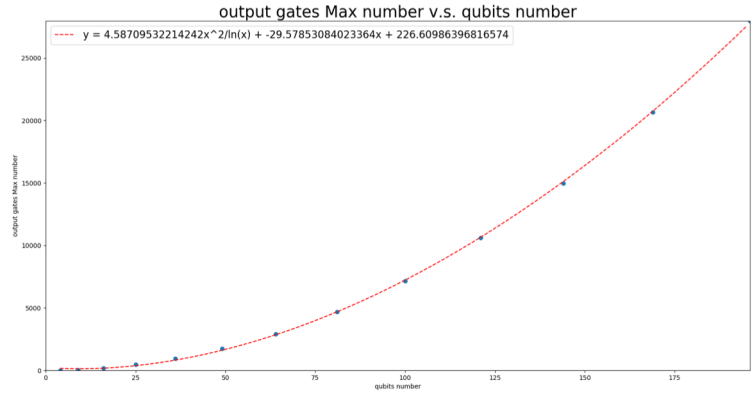








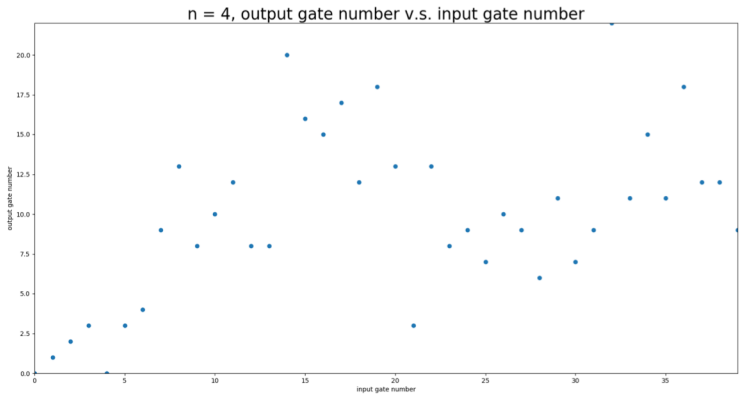


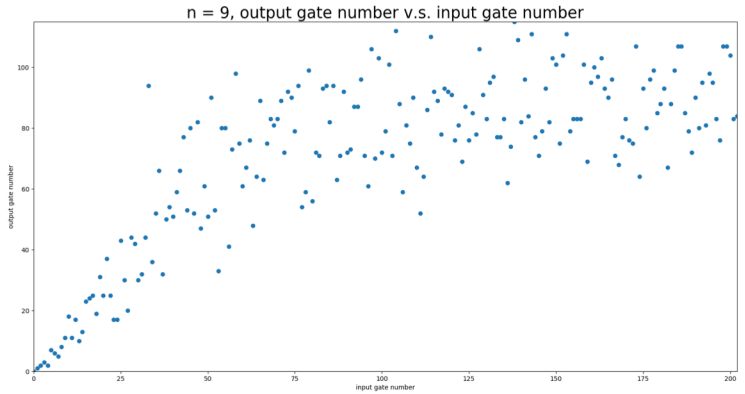


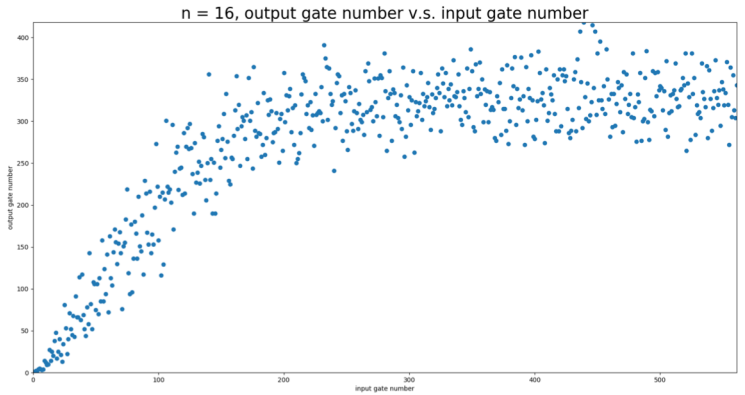
---- y = 4.58709532214242x^2/ln(x) + -29.57853084023364x + 226.60986396816574

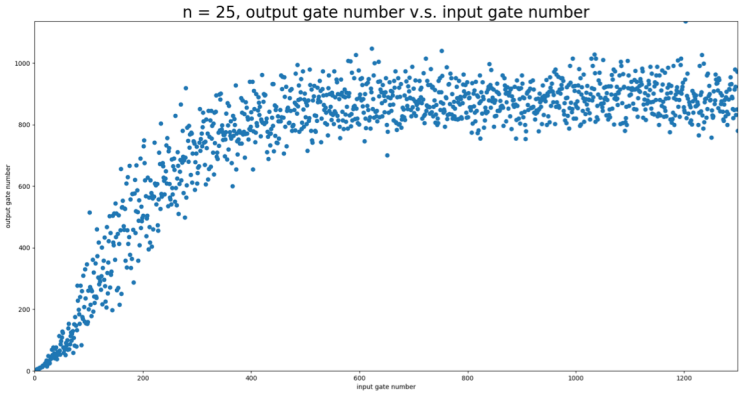
- Test circuit is generated by randomly generate C-NOT.

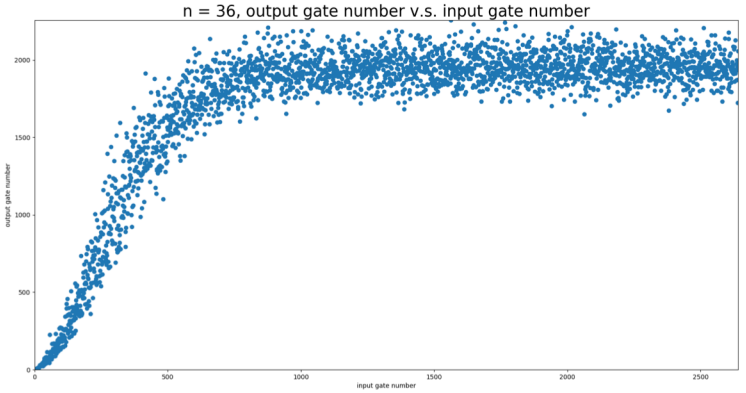
Experimental Results with connectivity

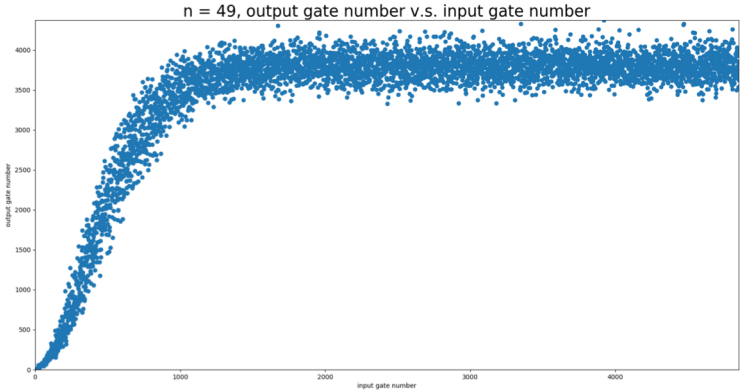






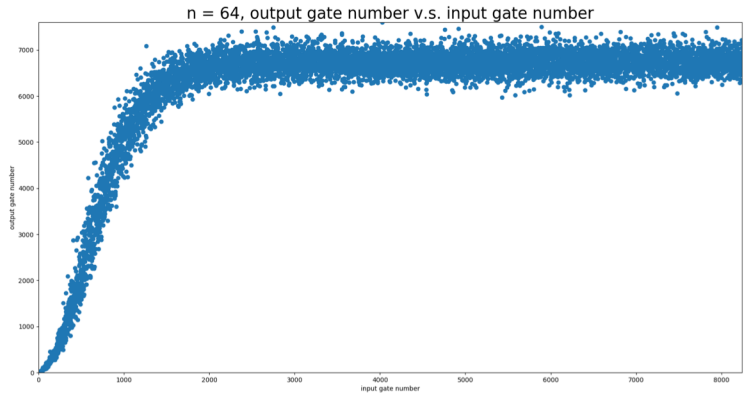


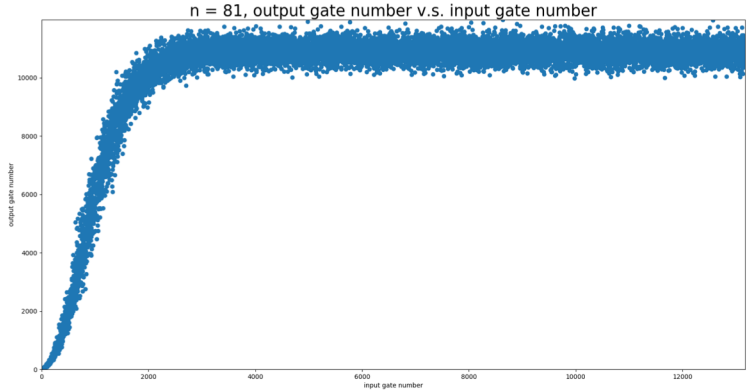


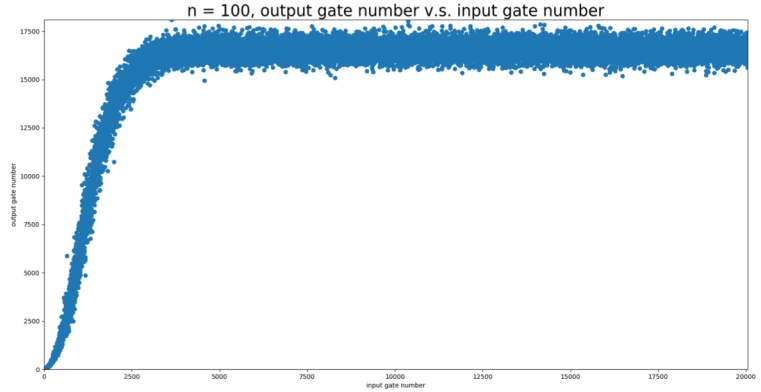


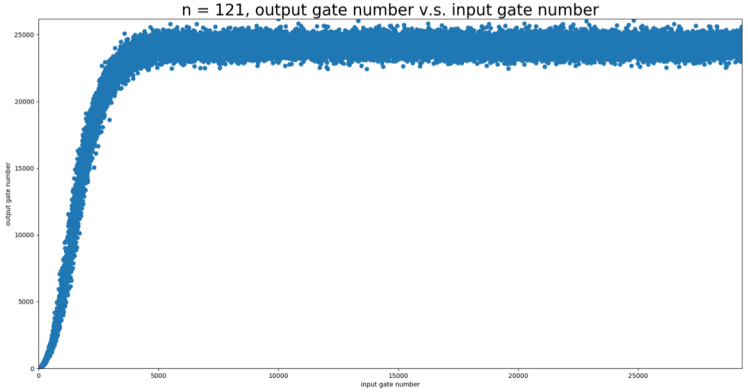
- Test circuit is generated by randomly generate C-NOT.

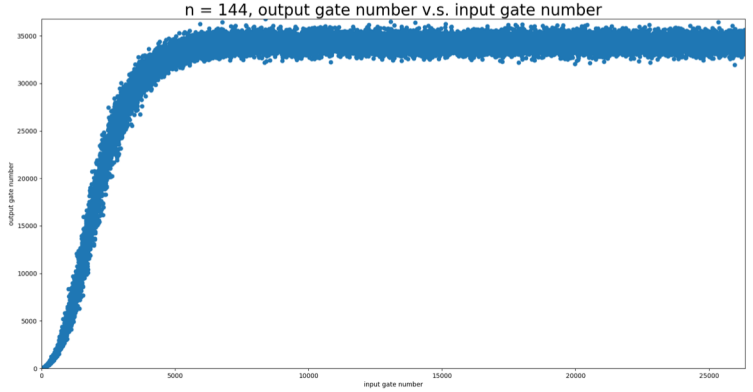
Experimental Results with connectivity

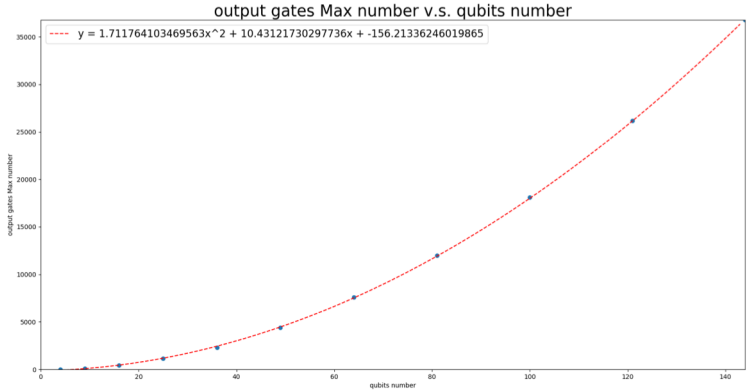








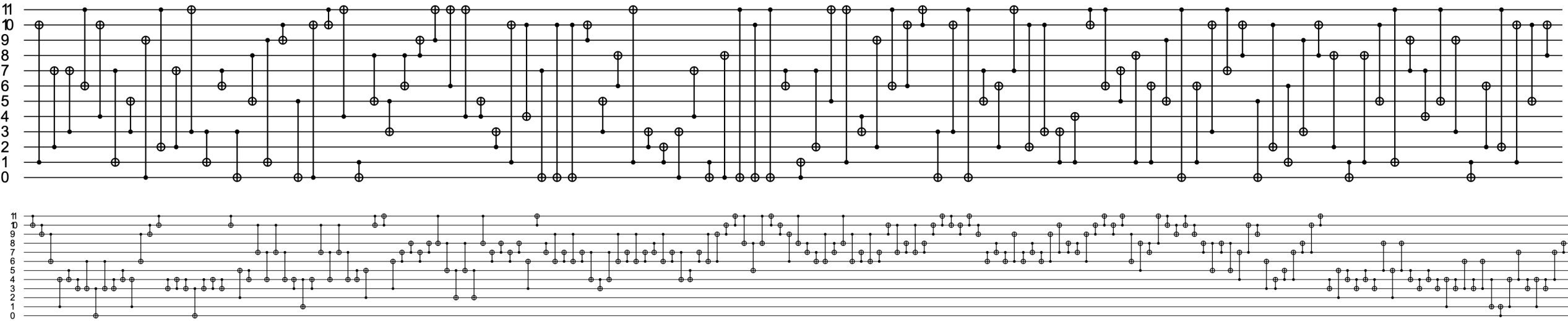


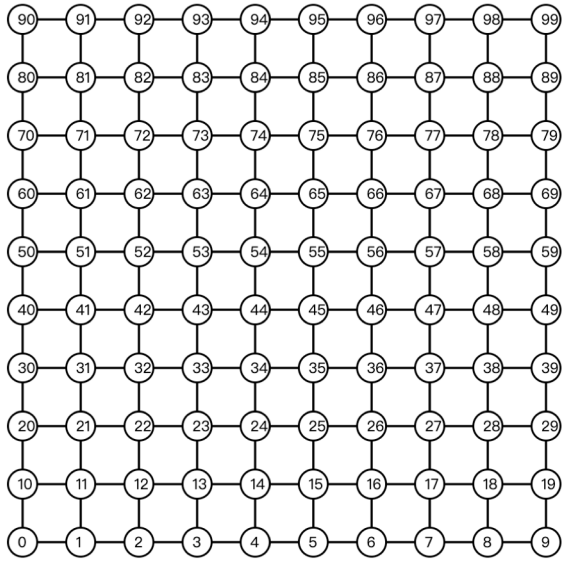


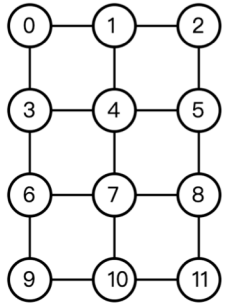
---- y = 1.711764103469563x^2 + 10.43121730297736x + -156.21336246019865

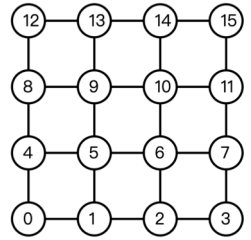
- Test circuit is generated by randomly generate C-NOT.

Visualization(python)









Outline

- Introduction and previous work
- Implementation and Contribution
- Experimental Results
- Observation and Discussion

Observation and Discussion

- Observation
  - These two method both perform bad at simple cases. Maybe it is because when focus at a single column, the other columns' information is lost.
  - The second method says that partitioning has bad performance. Maybe it is because of the increasing number of Steiner trees.
- Future work
  - Work on connectivity graph that is not rectangular.
  - Given a matrix and a connectivity graph architecture, how to place the qubits ?

Thank You!